### **Reading**

This material is covered in Chapter 12 of book.

### **File Class**

An **absolute file name** refers to the complete file name, eg. c:\myfolder\HelloWorld.java (Windows) or /users/jdoe/mystuff/HelloWorld.java (Unix).

Note c:\myfolder or /users/jdoe/mystuff are **directory path** of the files.

A **relative file name** is relative to the **current working directory** (where you are). If you are in /users/jdoe, the relative file name is mystuff/HelloWorld.java

File class provides an abstraction that hides all the machine-dependent complexities of dealing with files and directories.

It provides methods for accessing properties of, deleting, and renaming files / directories.

It does not provides methods to reading or writing file contents.

Use relative paths (eg. do not use "c:\myfolder\myfile.txt") and the forward slash directory separator (eg. "image/pic.gif") to ensure the program works both on Windows and Unix.

### **File Class Methods**

* exists() - check is the file object exists
* isDirectory() - checks if file object is directory
* isFile() - checks if file object is file
* canRead() - check is the file object exists and can be read
* canWrite() - check is the file object exists and be written
* lastModified() - date and time when last modified
* length() - size of file

Here is an example:

package com.foothill;

import java.io.File;

import java.util.Date;

public class TestProgram {

public static void main(String[] args) {

File file = new File("data.txt");

System.out.println("Does it exist? " + file.exists());

System.out.println("Is it a file? " + file.isFile());

System.out.println("Is it a directory? " + file.isDirectory());

System.out.println("Can it be read? " + file.canRead());

System.out.println("Can it be written? " + file.canWrite());

System.out.println("It was last modified on " + new Date(file.lastModified()));

System.out.println("The file has " + file.length() + " bytes");

}

}

The output is:

Does it exist? true

Is it a file? true

Is it a directory? false

Can it be read? true

Can it be written? true

It was last modified on Sun Oct 04 23:09:44 PDT 2020

The file has 39 bytes

### **File Input**

We can read text data from files using the Scanner class.

Scanner breaks input to tokens delimited by whitespace.

To read from the keyboard, create Scanner as:

Scanner input = new Scanner(System.in);

To read from a file, create Scanner as:

Scanner input = new Scanner(new File(filename));

Then the token reading methods, nextByte(), nextShort(), nextInt(), nextLong(), nextFloat() , and nextDouble() can be used to read token separated by **delimiters**.

The delimiter is whitespace by default, but can be changed with useDelimiter(String regex), where regex is a pattern for delimiters.

These methods (1) skip any delimiters, (2) read a String token ending with a delimiter and (3) automatically convert them to byte, short, int, long, float, double respectively.

If the String read can not be converted to the expected type, an InputMismatchException is raised.

The method next() can be use to perform no conversion at all.

The method nextLine() reads all remaining text (including delimiters) till the a line separator; the line separator is read but is not part of the String returned.

Consider the code:

Scanner input = new Scanner(new File("data.txt"));

int intVal = input.nextInt();

String txtVal = input.nextLine();

If data.txt contains the following:

12 345

intVal will contain the number 12, and txtVal will contain " 345" (note the first char is a space).

Alternatively, consider the code:

Scanner input = new Scanner(System.in);

int intVal = input.nextInt();

String txtVal = input.nextLine();

Suppose at the keyboard, you enter 12, press enter key, enter 345, press enter key.

Now, intVal will contain the number 12, and txtVal will contain "" (an empty string).

Here nextInt() reads 12 but stops before the delimiter (the enter key which is also the line separator). Then nextLine() reads till the line separator but returns the string before the line separator, which is "".

### **Testing Input**

The method hasNext() can be used to check if Scanner has more data to read.

Methods hasNextInt(), hasNextDouble(), etc. can use to check if the next token to be read is of a particular type.

For example:

int i;

if (input.hasNextInt())

i = input.nextInt();

else

System.out.println("Not an int");

### **Converting Strings**

* Use Integer.parseInt() to covert string to int

String iStr = "123";

int i = Integer.parseInt(intStr); // 123

* Use Double.parseDouble() to convert string to double

String dStr = "12.3";

double d = Double.parseDouble(dStr); // 12.3